UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/496,844	02/02/2000	Patrick Knebel	10971393-1	6757

22879 7590 12/24/2003

HEWLETT PACKARD COMPANY
P O BOX 272400, 3404 E. HARMONY ROAD
INTELLECTUAL PROPERTY ADMINISTRATION
FORT COLLINS, CO 80527-2400

EXAMINER	
HUISMAN, DAVID J	
ART UNIT	PAPER NUMBER
2183	

DATE MAILED: 12/24/2003 16

Please find below and/or attached an Office communication concerning this application or proceeding.

PTO-90C (Rev. 10/03)

Office Action Summary	Application No. 09/496,844	Applicant(s) KNEBEL ET AL.
	Examiner David J. Huisman	Art Unit 2183

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *29 October 2003*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1,3-5,7-15,18,19 and 21-24* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1,3-5,7-15,18,19 and 21-24* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *02 February 2000* is/are: a)☐ accepted or b)☒ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. §§ 119 and 120

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

13)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application) since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.

a) ☐ The translation of the foreign language provisional application has been received.

14)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121 since a specific reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)	4) ☐ Interview Summary (PTO-413) Paper No(s). _____ .
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) ☐ Notice of Informal Patent Application (PTO-152)
3) ☐ Information Disclosure Statement(s) (PTO-1449) Paper No(s) _____ .	6) ☐ Other: .

DETAILED ACTION

1. Claims 1, 3-5, 7-15, 18, 19, and 21-24 have been examined.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received and placed of record in the file: #14. RCE as received on 10/29/2003 and #15. Pre-Amendment "D" as received on 10/29/2003.

Specification

3. The title of the invention is not descriptive. A new title is required that is clearly indicative of the invention to which the claims are directed.

4. The disclosure is objected to because of the following informalities: On page 9, lines 9-12, of the originally filed specification, the applicant should include the Serial/Patent No. of the application incorporated by reference. In addition, the application in question is no longer copending, as it has been issued, and the title should be updated accordingly, i.e., the title should be changed to read --Method and Apparatus for Implementing Two Architectures in a Chip Using Bundles that Contain Microinstructions and Template Information--.

Appropriate correction is required.

5. The specification is objected to as failing to provide proper antecedent basis for the claimed subject matter. See 37 CFR 1.75(d)(1) and MPEP § 608.01(o). Correction of the following is required: The examiner has been unable to find support for lines 3-9 of claim 23 and lines 9-16 of claim 24. The specification does not provide a teaching of preventing parallel

issue of at least two microinstructions <u>with any prior microinstruction</u>, if such parallel issue would make it impossible to issue the at least two microinstructions in parallel. Without such a teaching, it is not clear to the examiner how one can prevent parallel issue of the at least two microinstructions with any prior microinstruction because a "prior microinstruction" implies that it was issued at a different time than the at least two microinstructions. Therefore, it is not clear how a prior instruction (i.e., previously issued) can be issued in parallel with subsequent microinstructions.

Drawings

6. The drawings are objected to under 37 CFR 1.83(a). The drawings must show every feature of the invention specified in the claims. Therefore, the prevention of parallel issue with any prior microinstruction (specified in new claims 23 and 24) must be shown or the feature(s) canceled from the claim(s). No new matter should be entered.

A proposed drawing correction or corrected drawings are required in reply to the Office action to avoid abandonment of the application. The objection to the drawings will not be held in abeyance.

Claim Objections

7. Claim 10 is objected to because of the following informalities: For increased clarity, it is recommended that the applicant replace "SSE" with what it actually represents. Appropriate correction is required.

8. Claim 18 is dependent on a cancelled claim (17). Appropriate correction is required. For purposes of this examination, claim 18 will be viewed as being dependent on claim 12.

9. Claim 18 is objected to because of the following informalities: Please replace "an Streaming" with --a Streaming-- in line 3. Appropriate correction is required.

10. Claim 21 is objected to because of the following informalities: Please replace "an Streaming" with --a Streaming-- in line 5. Appropriate correction is required.

11. Claim 22 is objected to because of the following informalities: Please replace "an Streaming" with --a Streaming-- in line 4. Appropriate correction is required.

Claim Rejections - 35 USC § 112

12. The following is a quotation of the first paragraph of 35 U.S.C. 112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

13. Claims 23 and 24 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the written description requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention. The specification does not provide a teaching of preventing parallel issue of at least two microinstructions <u>with any prior microinstruction</u>, if such parallel issue would make it impossible to issue the at least two microinstructions in parallel. Without such a teaching, it is not clear to the examiner how one can prevent parallel issue of the at least two microinstructions with any prior microinstruction because a "prior microinstruction" implies that it was issued at a

different time than the at least two microinstructions. Therefore, it is not clear how a prior instruction (i.e., previously issued) can be issued in parallel with subsequent microinstructions.

14. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

15. Claims 23 and 24 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. More specifically, it is not clear to the examiner how one can prevent parallel issue of the at least two microinstructions with any prior microinstruction because a "prior microinstruction" implies that it was issued at a different time than the at least two microinstructions. Therefore, it is not clear how a prior instruction (i.e., previously issued) can be issued in parallel with subsequent microinstructions. In addition, it is not clear how such a parallel issue would make it impossible to issue the at least two microinstructions in parallel. For example, if such a parallel issue includes issuing 3 microinstructions, then how can issuing 3 microinstructions in parallel make it impossible to issue 2 microinstructions in parallel?

Claim Rejections - 35 USC § 103

16. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

17. Claims 1, 3, 4, 7, 8, 12-15, and 19 are rejected under 35 U.S.C. 103(a) as being unpatentable over Col et al., U.S. Patent No. 6,330,657 B1 (as applied in the previous Office Action and herein referred to as Col) in view of Nakajima, U.S. Patent No. 5,537,561, in view of Shang et al., U.S. Patent No. 5,764,971 (as applied in the previous Office Action and herein referred to as Shang), and further in view of Song, U.S. Patent No. 5,546,599.

18. Referring to claim 1, Col has taught a method for processing software instructions comprising:

a) decomposing a macroinstruction into a plurality of microinstructions. See Fig.4, steps 402 and 404.

b) forcing the parallel issue of at least two of the plurality of microinstructions simultaneously. See column 3, lines 52-56. Col has not taught forcing the parallel issue regardless of conflict checking. However, Nakajima has taught such a concept. See Fig.12(a) and Fig.12(b). Note in Fig.12(a) that the two instructions in storage 60 are to be issued in parallel to pipelines 80 and 90. It should be noted that both registers write to the same destination (froo), i.e. a WAW hazard. Nakajima's system merely detects this conflict and adds tag information to each instruction, as shown in Fig.12(b). These tags are used to correct the conflict at a later point. However, as seen from Fig.12(b), the conflicting instructions are still issued in parallel. A description of this process is given in columns 11 and 12. The reason this is done is to realize parallel processing with a small amount of hardware without deteriorating processing efficiency though data dependencies are secured. See column 13, lines 33-38. This hardware reduction comes from the lack of need for waiting buffers as described in column 2, lines 46-51. And one of ordinary skill in the art would have realized that the more operations that are issued (and executed in parallel, the faster the system). Therefore, in order to realize parallel processing (which results in higher throughput) while reducing hardware and recognizing dependencies, it would have been obvious to one of ordinary skill in the art at the time of the invention to force a parallel issue regardless of conflict checking.

c) executing the at least two microinstructions simultaneously, in lockstep using functional units in a floating-point unit. See column 3, lines 31-35, and Fig.6 (note in cycle 7 that two microinstructions are executed in parallel). Furthermore, note from column 20, lines 32-38 and note that this parallel execution can occur using multiple floating-point functional units.

d) Col has not explicitly taught:

> d1) determining whether an exception occurs in any of the microinstructions, before writing results of the executing to result registers, wherein the determining step is performed prior to any writing step.
>
> d2) if an exception occurs in any of the microinstructions, canceling all of the microinstructions and preventing the results of the executing from being written to the result registers.
>
> d3) if no exception occurs in any of the microinstructions, writing the results of the executing to the result registers.

However, exceptions and the advantages of detecting exceptions are well known, accepted, and expected to occur in the art. In general, an exception is an interruption to the normal flow of program control, caused by the program itself or by executing an illegal instruction. Shang has taught a system in which macroinstructions are translated into a plurality of microinstructions and if an exception is detected within any one of those microinstructions, then the rest of the microinstructions are cancelled and results are not written to the result registers. Otherwise, if no exception is detected, then the results of the microinstructions are written to the result registers. See Fig.6 and note that if exceptions (interrupts) have been detected in at least one of the microinstructions at step 104, then the system is flushed (cancellation of each microinstruction) at step 108. If no exceptions were detected at step 104, then the results are written to the result registers at step 106. From this flowchart it can be seen that the determining of an exception occurs before the final writing step (step 106). The final writing step is "any writing step" as claimed by applicant. More specifically, the examiner is reading the claim as saying "the determining step is performed prior to a final writing step" because a final writing step is any writing step. In Col's system, since a macroinstruction is broken up into microinstructions, an exception in a single microinstruction would mean that an exception has occurred in the overall macroinstruction, and therefore, all of the microinstructions that represent a single macroinstruction, should not be able to change the state of the system. An advantage of Shang's scheme would be to avoid having to undo the changes made by the undesired execution of a microinstruction that is part of a faulty macroinstruction. This will prevent a reduction in throughput in that the extra time required to perform an undo-operation would not be necessary. Therefore, in order to maximize the efficiency of the overall system, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Col's system such that it employs exception detection among microinstructions as taught by Shang.

d4) Shang has not taught that the method does not write any results to temporary registers. However, Song has taught detecting exceptions by instructions without any writing occurring to temporary storage. See column 15, lines 37-51. More specifically, instruction results are written directly to final storage so that precise interrupts and precise exceptions will be achieved. In addition, not writing to temporary storage results in the removal of an extra writing step, thereby allowing for faster execution. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Shang such that temporary storage is not written to when detecting an exception.

19. Referring to claim 3, Col in view of Nakajima in view of Shang and further in view of Song has taught a method as described in claim 1. Col has further taught that the microinstructions are executed on separate execution units, but appear as though they were executed on a single execution unit. See column 20, lines 32-38. Note that multiple floating-point execution units are used per clock cycle in order to execute multiple microinstructions. For example, the microinstructions of cycle 7 in Fig.6 must be executed on different execution units if they are executed in parallel. Then the results of each microinstruction are written to the appropriate storage via store logic (component 420 of Fig.4). See column 17, lines 3-16. Note that the store logic retrieves all of the results from each microinstruction execution and writes the data to the appropriate place. Finally, from Fig.6 (cycle 7), it should be realized that the separate, but parallel, execution of LD T1,[BX] and ADD AX,T1, will produce a result that is expected for the ADD AX,[BX] macroinstruction.

20. Referring to claim 4, Col in view of Nakajima in view of Shang and further in view of Song has taught a method as described in claim 1. Col has further taught that all of the microinstructions are executed on the same clock cycle. See Fig.6, cycle 7, for instance.

21. Referring to claim 7, Col in view of Nakajima in view of Shang and further in view of Song has taught a method as described in claim 1. Furthermore, Col has taught that the system allows a single instruction to operate on multiple single-precision floating-point values. See column 3, lines 61-63.

22. Referring to claim 8, Col in view of Nakajima in view of Shang and further in view of Song has taught a method as described in claim 1. Col has not explicitly taught that a flag is updated based upon a result of the execution of the microinstructions. However, it is well known, accepted, and expected in the art that processors contain a status register. The status register contains bits that are set or cleared based on the result of an operation. Some of the more common flags are ones that indicate a result of zero, a negative number, and overflow. These flags can then be checked in conditional situations, such as branches. Therefore, it would have been obvious to one of ordinary skill in the art to update a flag based upon the result of the execution of the microinstructions.

23. Referring to claim 12, Col has taught a computer system comprising:

a) a processor comprising:

a1) a floating-point unit comprising a plurality of functional units adapted to execute microinstructions. See Fig.4, component 414, and column 20, lines 32-38.

a2) Col has not explicitly taught a computer system with a ROM. However, Official Notice is taken that ROMs are well known, accepted, and expected in the art. Processors contain Read-Only Memory to store essential software of the computer. Because it's non-volatile memory, ROM does not lose its contents when the power is turned off. Therefore, a ROM chip is used to store control programs for the computer, such as the bootstrap program (which tells the computer how to start and load the operating system) and other types of configuration information. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to provide some type of Rom in Col's system.

a3) a plurality of floating-point registers. See column 5, lines 49-52.

b) wherein the processor is configured to emulate an instruction set by:

b1) decomposing a macroinstruction into a plurality of microinstructions. See Fig.4, steps 402 and 404.

b2) forcing the parallel issue of at least two of the plurality microinstructions simultaneously to the functional units. See column 3, lines 52-56. Col has not taught forcing the parallel issue regardless of conflict checking. However, Nakajima has taught such a concept. See Fig.12(a) and Fig.12(b). Note in Fig.12(a) that the two instructions in storage 60 are to be issued in parallel to pipelines 80 and 90. It should be noted that both registers write to the same destination (froo), i.e. a WAW hazard. Nakajima's system merely detects this conflict and adds tag information to each instruction, as shown in Fig.12(b). These tags are used to correct the conflict at a later point. However, as seen from Fig.12(b), the conflicting instructions are still issued in parallel. A description of this process is given in columns 11 and 12. The reason this is done is to realize parallel processing with a small amount of hardware without deteriorating processing efficiency though data dependencies are secured. See column 13, lines 33-38. This hardware reduction comes from the lack of need for waiting buffers as described in column 2, lines 46-51. And one of ordinary skill in the art would have realized that the more operations that are issued (and executed in parallel, the faster the system). Therefore, in order to realize parallel processing (which results in higher throughput) while reducing hardware and recognizing dependencies, it would have been obvious to one of ordinary skill in the art at the time of the invention to force a parallel issue regardless of conflict checking.

b3) Col has not explicitly taught determining whether an exception occurs in any of the functional units wherein the determining step is performed prior to any setting step and the method does not set any temporary registers, setting result registers for results of each of the functional units only if no exception occurs in any of the functional units, and if an exception occurs in any of the microinstructions, canceling all of the microinstructions and preventing the setting of result registers for all of the functional units. However, exceptions and the advantages of detecting exceptions are well known, accepted, and expected to occur in the art. In general, an exception is an interruption to the normal flow of program control, caused by the program itself or by executing an illegal instruction. Shang has taught a system in which macroinstructions are translated into a plurality of microinstructions and if an exception is detected within any one of those microinstructions, then the rest of the microinstructions are cancelled and results are not written to the result registers. Otherwise, if no exception is detected, then the results of the microinstructions are written to the result registers. See Fig.6 and note that if exceptions (interrupts) have been detected in at least one of the microinstructions at step 104, then the system is flushed (cancellation of each microinstruction) at step 108. If no exceptions were detected at step 104, then the results are written to the result registers at step 106. From this flowchart it can be seen that the determining of an exception occurs before the final writing (setting) step (step 106). The final setting step is "any setting step" as claimed by applicant. More specifically, the examiner is reading the claim as saying "the determining step is performed prior to a final setting step" because a final setting step is any setting step. In Col's system, since a macroinstruction is broken up into microinstructions, an exception in a single microinstruction would mean that an exception has occurred in the overall macroinstruction, and therefore, all of the microinstructions that represent a single macroinstruction, should not be able to change the state of the system. An advantage of Shang's scheme would be to avoid having to undo the changes made by the undesired execution of a microinstruction that is part of a faulty macroinstruction. This will prevent a reduction in throughput in that the extra time required to perform an undo-operation would not be necessary. Therefore, in order to maximize the efficiency of the overall system, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Col's system such that it employs exception detection among microinstructions as taught by Shang.

b4) Shang has not taught that the method does not set any temporary registers. However, Song has taught detecting exceptions by instructions without setting temporary storage. See column 15, lines 37-51. More specifically, instruction results are written directly to final storage so that precise interrupts and precise exceptions will be achieved. In addition, not setting temporary storage results in the removal of an extra step (the setting step), thereby allowing for faster execution. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Shang such that temporary storage is not set when detecting an exception.

24. Referring to claim 13, Col in view of Nakajima in view of Shang and further in view of Song has taught a computer system as described in claim 12. Col has further taught that the processor is further configured to emulate the instruction set by executing all of the microinstructions. See column 3, lines 31-35, and Fig.6 (note in cycle 7 that two microinstructions are executed in parallel).

25. Referring to claim 14, Col in view of Nakajima in view of Shang and further in view of Song has taught a computer system as described in claim 13. Furthermore, it has been noted that the computer system of claim 14 performs the method of claim 3. Therefore, claim 14 is rejected for the same reasons set forth in the rejection of claim 3 above.

26. Referring to claim 15, Col in view of Nakajima in view of Shang and further in view of Song has taught a computer system as described in claim 14. Furthermore, it has been noted that the computer system of claim 15 performs the method of claim 8. Therefore, claim 15 is rejected for the same reasons set forth in the rejection of claim 8 above.

27. Referring to claim 19, Col in view of Nakajima in view of Shang and further in view of Song has taught a method as described in claim 1.

a) Col has further taught that the step of issuing comprises forcing the microinstructions to issue simultaneously, in lockstep with each other. See column 3, lines 52-56.

b) Col has not explicitly taught that the step of canceling comprises canceling all of the plurality of microinstructions without regard to the relative ages of the microinstructions and without using a backoff mechanism. However, recall from the rejection of claim 1 above, that it would have been an obvious modification to add exception detection functionality to Col's system (in view of Shang). This modification, as described above, would allow for the detection of an exception within a single microinstruction and if an exception is triggered, the other related microinstructions will be subject to cancellation. Furthermore, recall that if interrupts are detected within Shang's system, results are not written to result registers, and consequently, there is no need for a backoff mechanism to undo results that have been incorrectly written.

28. Claim 5 is rejected under 35 U.S.C. 103(a) as being unpatentable over Col in view of Nakajima in view of Shang in view of Song, as applied above, and further in view of Hennessy and Patterson, Computer Architecture - A Quantitative Approach, 2nd Edition, 1996 (as applied in the previous Office Action, mailed on 12/20/2002, and herein referred to as Hennessy).

29. Referring to claim 5, Col in view of Nakajima in view of Shang in view of Song has taught a method as described in claim 1. Col has further taught that his system can execute floating-point operations, which is supported in column 20, lines 32-38. Col has not explicitly taught that the microinstructions are executed over multiple clock cycles. However, it is well known, accepted, and expected in the art that floating-point operations can consume more than one clock cycle for execution purposes. Hennessy has disclosed this concept on page 187. Hennessy has also shown a pipeline that accommodates floating-point execution through multiple execution stages. See Fig.3.44 and Fig.3.45 on page 190. Since it has been disclosed by Hennessy that a floating-point operation takes multiple instruction execution cycles, it follows that it would have been obvious to one of ordinary skill in the art at the time of the invention to use a pipelined floating-point execution unit with multiple execution stages if floating-point operations are desired.

30. Claims 10, 11, 21, and 22 are rejected under 35 U.S.C. 103(a) as being unpatentable over Col in view of Nakajima in view of Shang in view of Song, as applied above, and further in view of Intel ®, Intel ® Architecture Optimization Reference Manual, 1998-1999 (as applied in the previous Office Action and herein referred to as Intel).

31. Referring to claim 10, Col has taught a method for processing software instructions comprising:

a) providing two microinstructions to emulate a high-half and a low-half operation. See Fig.6 and note in cycle 7 that two microinstructions are executed in parallel. Col has not explicitly taught that the operation is an SSE operation. However, Intel has taught that SSE instructions are used to accelerate performance of applications regarding 3D geometry. See page 1-12. As a result, in order to increase system performance, it would have been obvious to one of ordinary skill in the art at the time of the invention to allow the system of Col to emulate SSE operations.

b) forcing the high-half and low-half operations to issue in parallel. See column 3, lines 52-56. Col has not taught forcing the parallel issue regardless of conflict checking. However, Nakajima has taught such a concept. See Fig.12(a) and Fig.12(b). Note in Fig.12(a) that the two instructions in storage 60 are to be issued in parallel to pipelines 80 and 90. It should be noted that both registers write to the same destination (froo), i.e. a WAW hazard. Nakajima's system merely detects this conflict and adds tag information to each instruction, as shown in Fig.12(b). These tags are used to correct the conflict at a later point. However, as seen from Fig.12(b), the conflicting instructions are still issued in parallel. A description of this process is given in columns 11 and 12. The reason this is done is to realize parallel processing with a small amount of hardware without deteriorating processing efficiency though data dependencies are secured.

See column 13, lines 33-38. This hardware reduction comes from the lack of need for waiting buffers as described in column 2, lines 46-51. And one of ordinary skill in the art would have realized that the more operations that are issued (and executed in parallel, the faster the system). Therefore, in order to realize parallel processing (which results in higher throughput) while reducing hardware and recognizing dependencies, it would have been obvious to one of ordinary skill in the art at the time of the invention to force a parallel issue regardless of conflict checking.

c) dispatching the high-half and low-half operations simultaneously to a first floating point unit and to a second floating point unit, respectively. See column 3, lines 52-56, and column 20, lines 32-38.

d) executing the high-half and low-half operations simultaneously, in lockstep. See column 3, lines 31-35, and Fig.6 (note in cycle 7 that two microinstructions are executed in parallel).

e) generating a signal from an emulator's hardware. Signals are inherently generated within a computer system. For instance, a clock signal is a basic signal that synchronizes the many different information-processing tasks assigned to the chip. Also, as instructions are fetched and decoded, signals are sent to the appropriate functional units in order to "specify" which operations are to be performed based on the type of instruction.

f) sending the signal to the first and second floating point functional units. Again, the appropriate signals would have to be supplied to the appropriate functional units in order to perform the desired operation.

g) Col has not explicitly taught determining whether an exception is taken in either the first or the second floating point unit, wherein the determining step is performed prior to any writing step and if the exception is taken in either the first or second floating point unit, preventing results from the high-half and low-half operations from being written to result registers, and canceling both the high-half and low-half operations. However, exceptions and the advantages of detecting exceptions are well known, accepted, and expected to occur in the art. In general, an exception is an interruption to the normal flow of program control, caused by the program itself or by executing an illegal instruction. Shang has taught a system in which macroinstructions are translated into a plurality of microinstructions and if an exception is detected within any one of those microinstructions, then the rest of the microinstructions are cancelled and results are not written to the result registers. Otherwise, if no exception is detected, then the results of the microinstructions are written to the result registers. See Fig.6 and note that if exceptions (interrupts) have been detected in at least one of the microinstructions at step 104, then the system is flushed (cancellation of each microinstruction) at step 108. If no exceptions were detected at step 104, then the results are written to the result registers at step 106. From this flowchart it can be seen that the determining of an exception occurs before the final writing step (step 106). The final writing step is "any writing step" as claimed by applicant. More specifically, the examiner is reading the claim as saying "the determining step is performed prior to a final writing step" because a final writing step is any writing step. In Col's system, since a macroinstruction is broken up into microinstructions, an exception in a single microinstruction would mean that an exception has occurred in the overall macroinstruction, and therefore, all of the microinstructions that represent a single macroinstruction, should not be able to change the state of the system. An advantage of Shang's scheme would be to avoid having to undo the changes made by the undesired execution of a microinstruction that is part of a faulty macroinstruction. This will prevent a reduction in throughput in that the extra time required to perform an undo-operation would not be necessary. Therefore, in order to maximize the efficiency of the overall system, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Col's system such that it employs exception detection among microinstructions as taught by Shang.

h) Shang has not taught that the method does not write any results to temporary registers. However, Song has taught detecting exceptions by instructions without any writing occurring to temporary storage. See column 15, lines 37-51. More specifically, instruction results are written directly to final storage so that precise interrupts and precise exceptions will be achieved. In addition, not writing to temporary storage results in the removal of an extra writing step, thereby allowing for faster execution. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Shang such that temporary storage is not written to when detecting an exception.

i) Col has not explicitly taught updating MXCSR flags based upon the results of the first and second floating point units. However, the general idea of a status register is well known, accepted, and expected in the art. The MXCSR register contains flags that are common to other processor status registers. These bits (flags) are set and cleared based on results from operations. For instance, if the result of an addition were zero, a flag indicating a zero-result would be set in the status register. Or, perhaps an overflow occurred. A flag in the status register would be set to specify that as well. Conditional statements such as branches then reference these flags in order to determine the direction of the program. Therefore, in order to provide a readable status of the processor so that programs (via branches) flow according to previously obtained results, it would have been obvious to one of ordinary skill in the art at the time of the invention to update flags that are found in the MXCSR register.

32. Referring to claim 11, Col in view of Nakajima in view of Shang in view of Song and further in view of Intel has taught a method as described in claim 10. Recall that it has been established that it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Col such that it can detect and correct exceptions in a manner taught by Shang. Note also that Shang the preventing and canceling steps are performed regardless of the relative ages of the microinstructions. See Fig.6.

33. Referring to claim 21, Col in view of Nakajima in view of Shang and further in view of Song has taught a method as described in claim 1 wherein the step of executing comprises executing using a plurality of functional units of a floating-point unit. Col in view of Nakajima in view of Shang in view of Song has not explicitly taught the emulation of Streaming Single Instruction Multiple-Data Extensions (SSE) instructions. However, as discussed above, Intel has taught that SSE instructions are used to accelerate performance of applications regarding 3D geometry. See page 1-12. As a result, in order to increase system performance, it would have been obvious to one of ordinary skill in the art at the time of the invention to allow for the emulation of SSE instructions within the system of Col. Furthermore, it is inherent that within computer systems, as instructions are fetched and decoded, signals are sent to the appropriate functional units in order to "specify" which operations are to be performed based on the type of instruction. Therefore, if Col's system included SSE instructions, as established above, Col would have inherently taught:

a) generating a signal via hardware that indicates that the functional units are emulating an SSE instruction and sending the signal to the functional units. Again, the appropriate signals would have to be supplied to the appropriate functional units in order to perform the desired operation.

b) Also, it is inherent that determining an exception would occur after the functional unit has received its signal. For instance, the only way an overflow exception could be detected is by checking the result of an operation, which would only be obtained subsequent to "telling" the functional unit which operation to perform.

34. Referring to claim 22, Col in view of Nakajima in view of Shang in view of Song has taught a system as described in claim 12. Furthermore, it has been noted that the system of claim 22 performs the method of claim 21. Therefore, claim 22 is rejected for the same reasons set forth in the rejection of claim 21.

35. Claim 9 is rejected under 35 U.S.C. 103(a) as being unpatentable over Col in view of Nakajima in view of Shang in view of Song, as applied above, and further in view of Phillips et al., U.S. Patent No. 6,038,652 (as applied in the previous Office Action and herein referred to as Philips).

36. Referring to claim 9, Col in view of Nakajima in view of Shang in view of Song has taught a method as described in claim 1.

a) Col has not explicitly taught that if an unmasked exception occurs, canceling the execution of all of the plurality of microinstructions, without regard to the relative ages of each of the plurality of microinstructions, and invoking a microcode handler. However, recall from the rejection of claim 1 above, that it would have been an obvious modification to add exception detection functionality to Col's system (in view of Shang). This modification, as described above, would allow for the detection of an exception within a single microinstruction and if an exception is triggered, the other related microinstructions would be subject to cancellation. In addition, Shang has taught that the triggering of an exception will result in the invocation of a microcode handler (referred to as an interrupt service routine). See column 11, lines 21-25. In general, the handler is invoked in order to correct the cause and effects of the exception and allow the processor to continue execution. Therefore, in order to correctly service an exception, it would have been obvious to one of ordinary skill in the art at the time of the invention to implement a microcode handler, which must be invoked upon exception detection.

b) Col in view of Shang has not explicitly taught updating at least one exception flag (when an unmasked exception occurs) by independently generating a logical OR of exceptions for a plurality of functional units. However, Phillips has taught the concept of simultaneously checking SIMD elements for exceptions and combining each individual exception into an overall exception. See FIG.2. Furthermore, the combining element (230) in FIG.2 can be implemented as an OR gate that generates a flag (240) used to specify whether or not an exception has occurred. See column 3, lines 60-63. A person of ordinary skill in the art would have recognized that the concept of Phillips would be applicable in Col's system in order to check for exceptions during the parallel execution of microinstructions. In a SIMD processor (as taught by Col), the overhead incurred to process the many possible exceptions generated by SIMD elements may be expensive and lead to degradation in performance. The system of Philips provides an efficient technique to report exceptions occurring in computing complex functions on a SIMD machine. An advantage to this scheme is that since an exception flag is produced according to a parallel execution of microinstructions as opposed to a serial execution of microinstructions, the processor will be able to detect an exception sooner and therefore sooner make the determination that the involved microinstructions should be cancelled. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to update at least one exception flag in Col's system based on the exception check for a plurality of microinstructions.

37. Claim 18 is rejected under 35 U.S.C. 103(a) as being unpatentable over Col in view of Nakajima in view of Shang in view of Song and further in view of Intel, as applied above, and further in view of Makineni et al., U.S. Patent No 6,321,327 B1 (herein referred to as Makineni).

38. Referring to claim 18, Col in view of Nakajima in view of Shang in view of Song has taught a computer system as described in claim 12. Col has further taught the general use of SIMD instructions, which is the format used by SSE instructions. See column 3, lines 61-63. However, Col has not disclosed the specific use of Streaming Single Instruction Multiple-Data Extensions (SSE) instructions. Intel has taught that SSE instructions are used to accelerate performance of applications regarding 3D geometry. See page 1-12. As a result, in order to increase system performance, it would have been obvious to one of ordinary skill in the art at the time of the invention to implement an SSE instruction set within the system of Col. In addition, Col has taught that the SIMD execution units perform operations (such as an add) on multiple operands from a first SIMD register with corresponding multiple operands from a second SIMD register. See column 12, lines 1-8. Col has not explicitly taught the use of two 82-bit floating point registers for emulating four 32-bit single-precision floating-point values in an SSE register. However, Makineni has taught the use of 82-bit registers to hold 32-bit single precision floating-

point numbers. See Fig.2B and Fig.3. Since SIMD operations involve performing a single operation on multiple pairs of data elements, it follows that multiple pairs of operands must be available to the SIMD execution unit. A person of ordinary skill in the art would have recognized that by implementing 82-bit registers, multiple pairs of operands would be supplied to a SIMD execution unit by using multiple registers. In addition, by packing more than one data element into a single register, the amount of addressable registers could be decreased, resulting in less wires used for addressing purposes. Finally, each of the standard IEEE floating-point formats can be specified through the use of a single 82-bit floating point register, allowing the system to operate on different-precision operands depending on the situation. See the floating-point standards on page A-13 of Hennessy and note Fig.2A of Makineni shows a double-extended precision floating-point number. Therefore, in order to decrease the amount of hardware, while assuring the system has the capability of processing a wide variety of floating-point numbers, it would have been obvious to one of ordinary skill in the art to use two 82-bit floating point registers for emulating four 32-bit single-precision floating-point values in an SSE register.

39. **Claims 23 and 24 are rejected under 35 U.S.C. 103(a) as being unpatentable over Col in view of Nakajima and further in view of Shang, as applied above.**

40. Referring to claim 23, Col has taught a method for processing software instructions comprising:

a) decomposing a macroinstruction into a plurality of microinstructions. See Fig.4, steps 402 and 404.

b) determining whether at least two of the plurality of microinstructions are required to execute simultaneously (see column 3, lines 52-56, and Fig.7), and if so:

c) Col has not taught preventing parallel issue of the at least two microinstructions with any prior microinstruction, if such parallel issue would make it impossible to issue the at least two microinstructions in parallel and forcing the at least two of the plurality of microinstructions to issue in parallel regardless of conflict checking. However, Nakajima has taught such a concept. See Fig.12(a) and note that the at least two microinstructions (fmul and fadd) are not issued in parallel with any prior microinstruction (Ld), i.e., the parallel issue is prevented. In this situation it is clear that not all instructions will be issued in parallel because there are only two pipelines and two pipelines cannot execute three instructions in parallel (i.e., there are insufficient resources to allow for such a parallel issue). Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to prevent the parallel issue of the at least two microinstructions with any prior microinstruction if there are in sufficient resources. And, looking at Fig.5 of Col, issuing 3 instructions in parallel is impossible since there are only 2 issue registers (525 and 526), i.e., there are insufficient resources. In addition, Nakajima has taught forcing the parallel issue regardless of conflict checking. Again, see Fig.12(a) and Fig.12(b). Note in Fig.12(a) that the two instructions in storage 60 are to be issued in parallel to pipelines 80 and 90. It should be noted that both registers write to the same destination (froo), i.e. a WAW hazard. Nakajima's system merely detects this conflict and adds tag information to each instruction, as shown in Fig.12(b). These tags are used to correct the conflict at a later point. However, as seen from Fig.12(b), the conflicting instructions are still issued in parallel. A description of this process is given in columns 11 and 12. The reason this is done is to realize parallel processing with a small amount of hardware without deteriorating processing efficiency though data dependencies are secured. See column 13, lines 33-38. This hardware reduction comes from the lack of need for waiting buffers as described in column 2, lines 46-51. And one of ordinary skill in the art would have realized that the more operations that are issued (and executed in parallel, the faster the system). Therefore, in order to realize parallel processing (which results in higher throughput) while reducing hardware and recognizing dependencies, it would have been obvious to one of ordinary skill in the art at the time of the invention to force a parallel issue regardless of conflict checking.

d) Col has further taught executing the at least two microinstructions simultaneously, in lockstep using functional units in a floating-point unit. See column 3, lines 31-35, and Fig.6 (note in cycle 7 that two microinstructions are executed in parallel). Furthermore, note from column 20, lines 32-38 and note that this parallel execution can occur using multiple floating-point functional units.

e) Col has not explicitly taught:

d1) determining whether an exception occurs in any of the microinstructions, before writing results of the executing to result registers.

d2) if an exception occurs in any of the microinstructions, canceling all of the microinstructions and preventing the results of the executing from being written to the result registers.

d3) if no exception occurs in any of the microinstructions, writing the results of the executing to the result registers.

However, exceptions and the advantages of detecting exceptions are well known, accepted, and expected to occur in the art. In general, an exception is an interruption to the normal flow of program control, caused by the program itself or by executing an illegal instruction. Shang has taught a system in which macroinstructions are translated into a plurality of microinstructions and if an exception is detected within any one of those microinstructions, then the rest of the microinstructions are cancelled and results are not written to the result registers. Otherwise, if no exception is detected, then the results of the microinstructions are written to the result registers. See Fig.6 and note that if exceptions (interrupts) have been detected in at least one of the microinstructions at step 104, then the system is flushed (cancellation of each microinstruction) at step 108. If no exceptions were detected at step 104, then the results are written to the result registers at step 106. In Col's system, since a macroinstruction is broken up into microinstructions, an exception in a single microinstruction would mean that an exception has occurred in the overall macroinstruction, and therefore, all of the microinstructions that represent a single macroinstruction, should not be able to change the state of the system. An advantage of Shang's scheme would be to avoid having to undo the changes made by the undesired execution of a microinstruction that is part of a faulty macroinstruction. This will prevent a reduction in throughput in that the extra time required to perform an undo-operation would not be necessary. Therefore, in order to maximize the efficiency of the overall system, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Col's system such that it employs exception detection among microinstructions as taught by Shang.

41. Referring to claim 24, Col has taught a computer system comprising:

a) a processor comprising:

a1) a floating-point unit comprising a plurality of functional units adapted to execute microinstructions. See Fig.4, component 414, and column 20, lines 32-38.

a2) Col has not explicitly taught a computer system with a ROM. However, Official Notice is taken that ROMs are well known, accepted, and expected in the art. Processors contain Read-Only Memory to store essential software of the computer. Because it's non-volatile memory, ROM does not lose its contents when the power is turned off. Therefore, a ROM chip is used to store control programs for the computer, such as the bootstrap program (which tells the computer how to start and load the operating system) and other types of configuration information. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to provide some type of Rom in Col's system.

a3) a plurality of floating-point registers. See column 5, lines 49-52.

b) wherein the processor is configured to emulate an instruction set by:

b1) decomposing a macroinstruction into a plurality of microinstructions. See Fig.4, steps 402 and 404.

b2) determining whether at least two of the plurality of microinstructions are required to execute simultaneously (see column 3, lines 52-56, and Fig.7), and if so:

c) Col has not taught preventing parallel issue of the at least two microinstructions with any prior microinstruction, if such parallel issue would make it impossible to issue the at least two microinstructions in parallel and forcing the at least two of the plurality of microinstructions to issue in parallel regardless of conflict checking. However, Nakajima has taught such a concept. See Fig.12(a) and note that the at least two microinstructions (fmul and fadd) are not issued in parallel with any prior microinstruction (Ld), i.e., the parallel issue is prevented. In this situation it is clear that not all instructions will be issued in parallel because there are only two pipelines and two pipelines cannot execute three instructions in parallel (i.e., there are insufficient resources to allow for such a parallel issue). Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to prevent the parallel issue of the at least two microinstructions with any prior microinstruction if there are in sufficient resources. And, looking at Fig.5 of Col, issuing 3 instructions in parallel is impossible since there are only 2 issue registers (525 and 526), i.e., there are insufficient resources. In addition, Nakajima has taught forcing the parallel issue regardless of conflict checking. Again, see Fig.12(a) and Fig.12(b). Note in Fig.12(a) that the two instructions in storage 60 are to be issued in parallel to pipelines 80 and 90. It should be noted that both registers write to the same destination (froo), i.e. a WAW hazard. Nakajima's system merely detects this conflict and adds tag information to each instruction, as shown in Fig.12(b). These tags are used to correct the conflict at a later point. However, as seen from Fig.12(b), the conflicting instructions are still issued in parallel. A description of this process is given in columns 11 and 12. The reason this is done is to realize parallel processing with a small amount of hardware without deteriorating processing efficiency though data dependencies are secured. See column 13, lines 33-38. This hardware reduction comes from the lack of need for waiting buffers as described in column 2, lines 46-51. And one of ordinary skill in the art would have realized that the more operations that are issued (and executed in parallel, the faster the system). Therefore, in order to realize parallel processing (which results in higher throughput) while reducing hardware and recognizing dependencies, it would have been obvious to one of ordinary skill in the art at the time of the invention to force a parallel issue regardless of conflict checking.

b3) Col has not explicitly taught:

d1) determining whether an exception occurs in any of the functional units.

d2) setting result registers for results of each of the functional units only if no exception occurs in any of the functional units.

d3) if an exception occurs in any of the microinstructions, canceling all of the microinstructions and preventing the setting of result registers for all of the functional units.

However, exceptions and the advantages of detecting exceptions are well known, accepted, and expected to occur in the art. In general, an exception is an interruption to the normal flow of program control, caused by the program itself or by executing an illegal instruction. Shang has taught a system in which macroinstructions are translated into a plurality of microinstructions and if an exception is detected within any one of those microinstructions, then the rest of the microinstructions are cancelled and results are not written to the result registers. Otherwise, if no exception is detected, then the results of the microinstructions are written to the result registers. See Fig.6 and note that if exceptions (interrupts) have been detected in at least one of the microinstructions at step 104, then the system is flushed (cancellation of each microinstruction) at step 108. If no exceptions were detected at step 104, then the results are written to the result registers at step 106. From this flowchart it can be seen that the determining of an exception occurs before the final writing step (step 106). The final writing step is "any writing step" as claimed by applicant. More specifically, the examiner is reading the claim as saying "the determining step is performed prior to a final writing step" because a final writing step is any writing step. In Col's system, since a macroinstruction is broken up into microinstructions, an exception in a single microinstruction would mean that an exception has occurred in the overall macroinstruction, and therefore, all of the microinstructions that represent a single macroinstruction, should not be able to change the state of the system. An advantage of Shang's scheme would be to avoid having to undo the changes made by the undesired execution of a microinstruction that is part of a faulty macroinstruction. This will prevent a reduction in throughput in that the extra time required to perform an undo-operation would not be necessary. Therefore, in order to maximize the efficiency of the overall system, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Col's system such that it employs exception detection among microinstructions as taught by Shang.

Conclusion

42. Applicant's arguments have been considered but are moot in view of the new ground(s) of rejection.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to David J. Huisman whose telephone number is (703) 305-7811. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (703) 305-9712. The fax phone number for the organization where this application or proceeding is assigned is (703) 746-7239.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is (703) 305-3900.

DJH
David J. Huisman
December 12, 2003

[illegible]
RICHARD L. ELLIS
PRIMARY EXAMINER